Ian Hunt
Patrick McAfee
Christoffer Rosen

List of Assumptions:

- Elves can repeatedly have problems.
- Christmas only comes once a year.
- Elves will not start with a problem.
- Reindeer go to the tropics before the program is triggered.
- Elves will not have problems fixed while Santa is off doing Christmas, though they can still have problems during that time period.
- Numbers of elves and reindeer are immutable while the program is running.
- All reindeer must stay with Santa for all of Christmas before returning.
- Elves have no union, and are thus always working.
- Reindeer arrive very close to Christmas.
- Elves are semi-independent, and will only have their problems after a certain amount of time.
- Elves will not hang around waiting for their friends once their problem has been solved.
- Once three elves leave to get their problems fixed, no other elves will tag along once they've gone to Santa, and will wait for a new group of three.
- The exact elf to wake up Santa will be helped instantly.
- Santa will finish helping the current batch of Elves before hooking up the sleigh and doing Christmas.
- Santa's magic powers can have Christmas day last as many milliseconds as he needs.

Design Rationale:

Our design initially hinged on three main areas where concurrency was a concern, namely Elves, Reindeer, and Santa himself. We realized very quickly that Santa had very little functionality in and of himself, and his role was primarily one of structure and notification. He was the shared resource that Elves and Reindeer needed to access. As such, we decided to make Santa a simple object, not a thread. There was no reason for him to operate as a distinct thread. However, this meant that we would need to control access to Santa. To further this end, we had every call to Santa (save the call signifying Christmas) go through the synchronized method "awaken." Santa would then check the status of the workshop and take the appropriate action.

The second area of focus was the Elves. We wanted to keep a reference to all the elves with problems without adding extras and raising a case where Santa fixes the problems of elves he does not know about. We decided to utilize an arrayBlockingQueue from the Java library, as it would accept three elves, then would keep any others in a hidden "waiting queue" under the hood, adding them as room was freed in the queue. This structure was explicitly thread safe. When three elves are added to the queue, if there are three then the last Elf goes and wakes up Santa. Each Elf waits, synchronized on themselves, and Santa will wake their thread (except the one to awaken him, who gets instant problem solving). They just continue running, letting Santa do all the work.

The last major area was the reindeer. For this problem, we decided to utilize a cyclic barrier. Cyclic barrier proved to be a very natural choice, since it could take in a runnable which the last thread that arrived would execute. Per requirement, the last reindeer had to notify Santa

that all reindeer are accounted for, which is precisely what was achieved with the cyclic barrier. In addition, we used a CountDownLatch that would countdown once Santa had hooked up all the reindeer to the sled before the reindeer took off and and were in the state delivering. Of course, Santa will only hook up the reindeer once its Christmas, leaving that logic completely to Santa. The reindeer need only let Santa know that they are all waiting in the warming hut.

Race conditions:

During the initial testing phase, we found that reindeer were returning to the tropics while Christmas was still going on. This was tracked down to a misplaced line of code where they were released by the latch early. Switching two lines of code solved this problem.

Early diagnostic testing also revealed that Santa was finishing with elves before they were printing out that they had been fixed. While “under the hood,” the problem/fix cycle was continuing correctly, the print statements made it appear incorrect. Switching some statements around and rephrasing avoided unnecessary confusion and latches.

Lastly, there remains a case of two alternating race conditions, of which one is still in our project. There is an extremely rare case (which we have logically determined but never seen) where an elf can get it’s problem solved during Christmas by virtue of the “last elf need no notification.” However, when we attempted an alternate solution by implementing our own version of the array blocking queue to do notifications, we ran into a case where Santa would notify elves before they were waiting and thus the elf would wait without being awoken. As the second case occurs far more frequently than the first (which again, we have never actually seen), we opted for the first solution.

Test Cases:

Please see the attached test cases in the testcases folder. This contains text files of tests that include description (purpose) of the test as well as what configuration (constants) were used and the resulting output.